Accessing Microsoft Excel® Workbook Cell Attributes from within SAS® v9

Timothy J Harrington, Dataceutics, Inc. Pottstown, PA

ABSTRACT

Data in a Microsoft Excel worksheet is now readily accessible to SAS v9 using the IMPORT Procedure, and to SAS v9 and earlier using the DDE triplet. Although worksheet cell contents can be read into a SAS dataset, cell attributes such as bold, italics, underscore, strikethrough, and color are not, at least in present releases of SAS, readily accessible. This paper discusses and uses an example SAS program to show how cell attributes in Microsoft Excel can be determined by a SAS program reading the contents of an Excel worksheet.

INTRODUCTION

Situations often occur when reading data or text from Excel when the formatting of the cells, such as text color is important. An example is a list of items to be read into a SAS dataset, but when an item is no longer required it is marked with a strike-though instead of being deleted. This is often good practice for auditing purposes. Another example is where the color of the text is to be used to determine how the SAS system will process it.

EXAMPLE SITUATION

The example shown here is an Excel worksheet named **'*Study Datasets*'** containing the following list of SAS dataset IDs and names which may or may not be included in a proposed SAS library.

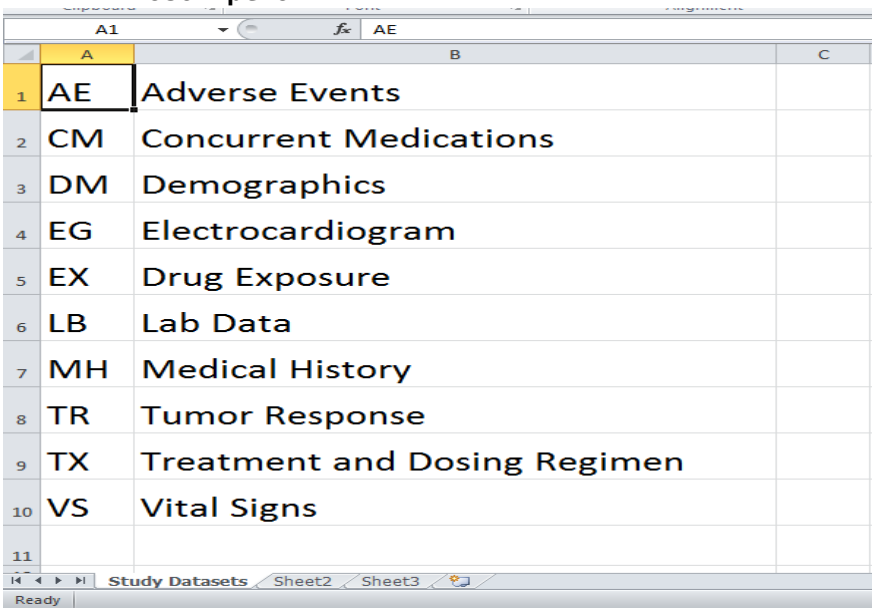

Figure 1. Worksheet Listing of Dataset IDs and Names

Over time decisions are made as to which datasets to include and which to exclude, but for auditing purposes, previously included but no longer required datasets cannot be deleted from the list, instead they are marked as excluded by using the strike-through attribute. In this example if the Electrocardiogram (EG) and Tumor Response (TR) datasets are to be dropped the worksheet would look like this:

```
Dataset Dataset
  ID    Description
```

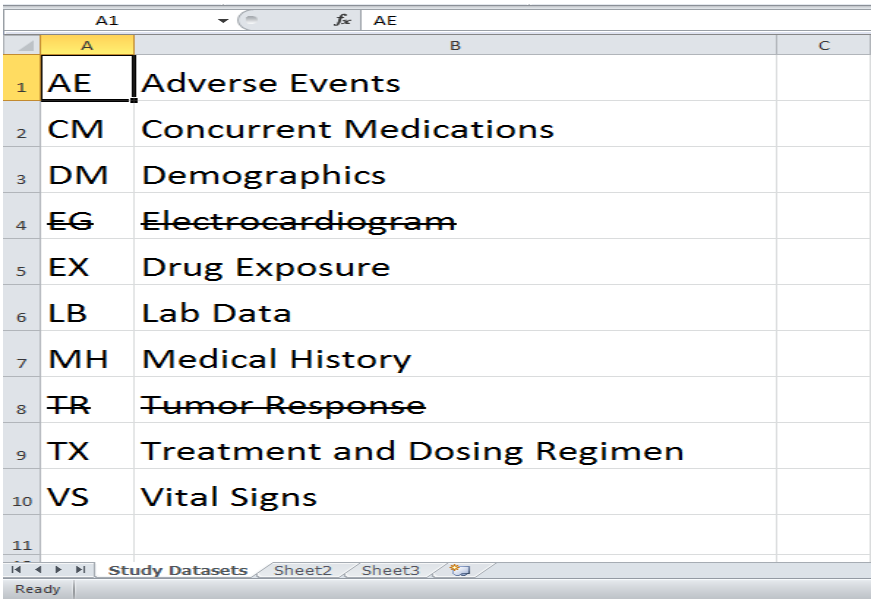

Figure 2. Same Sheet as Figure 1 but with Two of the Dataset IDs and Names Struck Out

The problem now is how to 'read' the strike-through, which is not a text character but an attribute of each cell containing the characters. Unfortunately there is no widely known way present versions of SAS are able to access attributes in MS Excel directly. However, attribute data is readily handled by Microsoft Visual Basic (VB) and VB commands can be issued from within a SAS DATA step using the DDE triplet. This means the struck-through rows of data can be identified and tagged with a text indicator (A unique character sequence such as '<XOUT>') in the first column (Column A). The data in this first column can then be reread with this tagging and be merged against the original data and the presence or absence of the tag indicates if the first column had the strike-through attribute or not.

The diagram overleaf shows the overall process. There are four parts to the SAS program:

1. Reading the original data into a SAS data set
2. Writing the VB commands to a VB macro and running the macro
3. Reading the first column of the data tagged by the VB macro into a SAS dataset
4. Merging the original and tagged SAS datasets to select the required observations

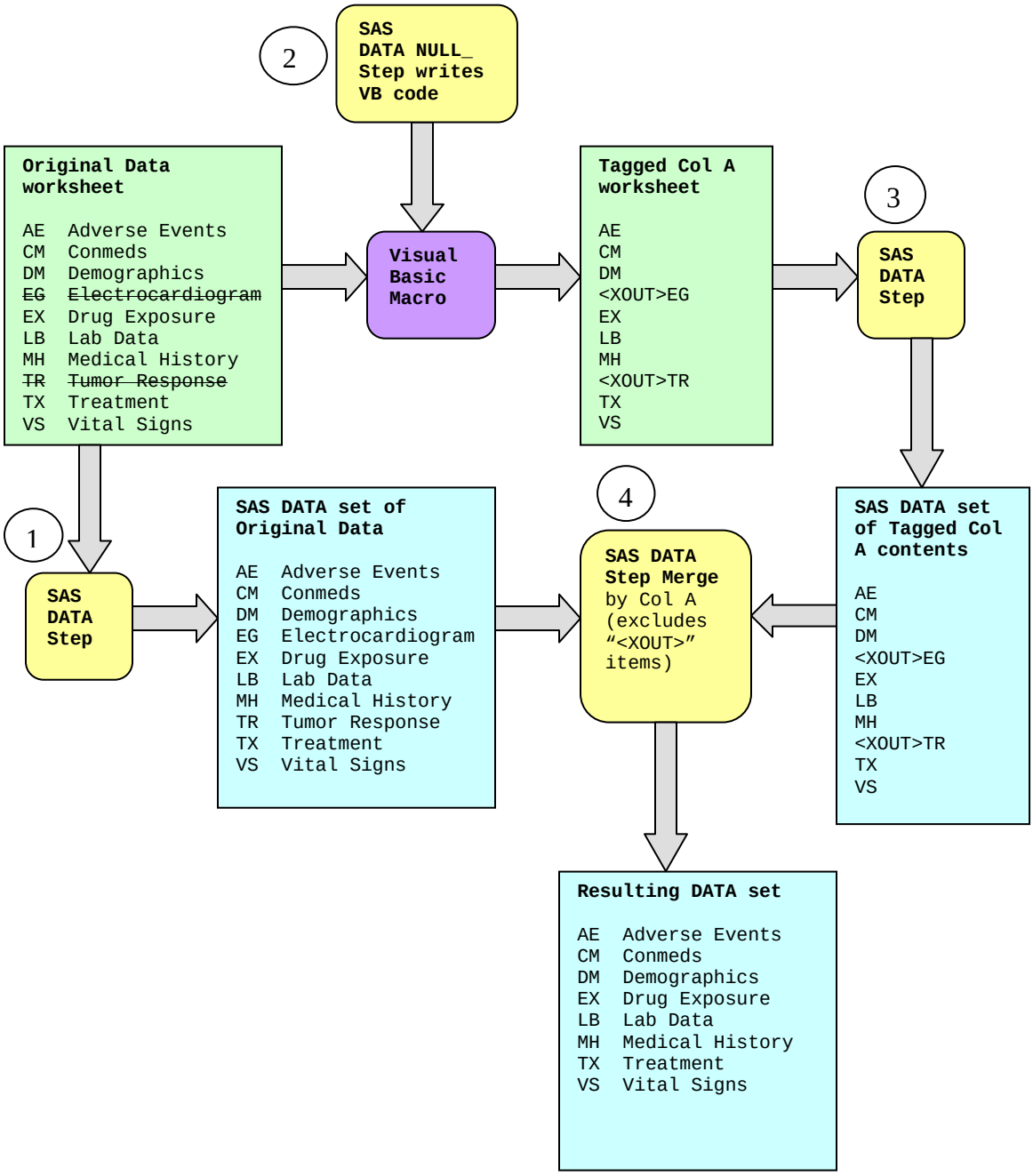

Figure 3. Overall Process Diagram

METHOD

Using DDE to access MS Excel is fairly common practice but it must be done carefully due to MS Excel and SAS running concurrently. The top-down design, including the SAS program steps from the above diagram is:

Run the Excel application from within SAS and define it as a DDE file

Read the original worksheet data into a SAS dataset (Step 1)

Create a new sheet in the opened workbook to hold the VB commands

Create a new sheet to hold the output from the VB code (tagged Column A)

Load the VB commands into the first sheet and execute them. The VB code will find the first character of the text in each of the cells of the first column (Column A) with the required attribute (strike-though in this example), mark each row containing that attribute with the unique character sequence, and then output to the second sheet (Step 2)

Read the second sheet containing the tagged Column A data including the marked character sequence into a SAS dataset. (Step 3)

Close the Excel workbook and application without saving, so no changes are made to the original workbook.

Merge the marked Column A dataset with the original data dataset by Column A. The observations with the tagged Column A values will not match and hence be dropped. (Step 4)

EXAMPLE SAS PROGRAM

The following program opens the Excel worksheet and performs the above functions, each section of code is explained in due course. Before running the program Excel should not already be running, the workbook should be read and write-accessible, and the data on the page to be read should all be displaying, not just a subset displayed from the column header, unless that specific subset is all that is needed. The program will re-open the Excel workbook exactly as it was previously closed.

The first step is to start the DDE application from within the SAS program by issuing the command to run Excel from the Windows command window. To do this without the user being prompted and to keep the command window minimized on the screen the applicable SAS options ***noxsync***, ***noxwait***, and ***xmin*** must be set. Then the command to run the Excel system can be issued using the SAS ***x call*** facility. Note: The full path of the location of excel.exe must be specified.

Having started the Excel system the next step is to open the DDE triplet as a SAS filename. A problem arises with synchronizing the execution of Excel and SAS concurrently, so the SAS execution must be temporarily suspended to allow the Excel system to complete its current task. This is achieved by using a DATA _NULL_ with a five second delay. This causes the message ‘SAS will awaken at <hh:mm:ss>’ to be shown on the screen while SAS is suspended. The ***filename*** statement then assigns the filename ***sas2xl*** to the Excel system.

```
options noxsync noxwait xmin;

x call "C:\Program Files\Microsoft Office\Office12\excel.exe";

%let delay=5;

data _null_;
  rc=sleep(&delay);
run;

filename sas2xl dde 'excel|system';

data _null_;
  rc=sleep(&delay);
run;
```

Now SAS is communicating with the MS Excel application, the next stage is to open the specific workbook, the name of which is stored in the macro variable ***mapwkbk***. This must be the complete directory path and file extension '.xlsx'. For example: 'C:/clinical studies/study a1/list of raw datasets.xlsx'. Note the need for the delay to synchronize the SAS and MS Excel applications before issuing the VB command to open the workbook.

```
data _null_;
  file sas2xl;
  rc=sleep(&delay);
  put '[open("' "&mapwkbk" '")]';
run;
```

Having opened the workbook the first task (Step 1 in the diagram) is to create a dataset containing the original data from the worksheet. This must be done before running the Visual Basic macro which will look for the strikethrough attribute because the macro will add marker tags to these data. As shown in the diagram above, the original data will later be merged with the marked observations after the macro has run. The macro variables ***maxrows*** and ***maxrecl*** must contain the maximum column number, in this case 2, and the maximum record length to be read, respectively. The ***filename*** statement specifies the sheet and the range of cells on that sheet to be read into the dataset, in this example the range is from row 1 and column 1 (top left corner cell) to ***&maxrows*** and column 2. The ***dlm*** parameter '09'x specifies the tab character as being the cell delimiter. (The ***notab*** in the ***filename*** statement is to allow the ***dlm*** option to be used either with the tab or any other character).

```
%let sheet_id=Study Datasets;
%let maxrows=10;
%let maxrecl=40;

filename origfile dde "excel|&sheet_id!r1c1:r&maxrows.c2" notab;

data original;
  length col_a col_b $&maxrecl;
  infile origfile dsd dlm='09'x missover pad lrecl=&maxrecl;
  input col_a $ col_b $;
run;

filename origfile clear;
```

Having read the original data into the dataset ***original***, the next task is to insert two new sheets into the ***mapwkbk*** workbook, one to contain the VB commands to look for the strike-through attribute (Step 2 in the diagram) and the other to contain the tagged Column A values output by the VB code.

```
data _null_;
  file sas2xl;
  put '[workbook.next()]';
  put '[workbook.insert(3)]';
run;
```

The first new sheet must now be opened as a DDE file for writing the VB commands to:

```
filename xlmacro dde "excel|macro1!r1c1:r99.c1" notab
  lrecl=&maxrecl;
```

The filename is ***xlmacro*** and the VB macro to be created will be called ***macro1*** (this is also the default name) and the code will occupy rows 1 onwards (up to 99 rows) in column 1. Only one column is needed for the VB code.

The sheet is now ready for the VB commands to be written to it and to be run. The DATA _NULL_ step is used with ***put*** statements to write the code to the sheet. The first line tells the ***put*** statements to write to the sheet defined above as ***xlmacro***. The ***put*** statements output the VB text as literal character strings and hence must be enclosed in quotation marks, double quotation marks are needed to enable the resolution of macro variables such as ***&sheet_id***, which contains the name of the sheet containing the input data. However, first, the attribute being sought, in this case 'strike-through' needs to be specified, as do the characters that will be used to mark a row as struck-out. This example code uses the macro variable ***attrcode*** to contain the cell attribute code, in this case 23, the 'strike-through'

attribute number. The most commonly used attribute numbers are listed at the end of this paper. The macro variable ***marked*** will contain the characters used to mark a row as struck-out.

```
%let attrcode=23;
%let marked='<XOUT>';

data _null_;
  file xlmacro;
  put '=set.name("Tag",!$b$1)';
  put '=formula("' "&marked"' ",Tag)';
  put '=set.name("OldValue",!$c$1)';
  put '=set.name("NewValue",!$b$2)';
  put '=for.cell("CurrentCell",' "'" "&sheet_id" "'"
        "!$a$1:$a$&maxrows,true)";
  put "=if(get.cell(&attrcode,CurrentCell),
        formula(get.cell(5,CurrentCell),OldValue),)";
  put '=formula("=concatenate(Tag,OldValue)",NewValue)';
  put "=if(get.cell(&attrcode,CurrentCell),
        formula(NewValue,CurrentCell),)";
  put '=next()';
  put '=halt(true)';
  put '!dde_flush';
  file sas2xl;
  put '[run("macro1!r1c1")]';
run;
```

The first ***put*** statement names the first blank cell as '**Tag**', the second statement then populates that cell, using the ***formula*** command with a character string '<XOUT>'. The '<' and '>' characters surrounding the 'XOUT' string are to ensure uniqueness in case these characters happen to be in the text. The next two ***put*** statements specify holding cells to contain the next row read (***OldValue***), and subsequently written with or without the '<XOUT>' text accordingly (***NewValue***). Now the first column of the input sheet ***&sheet_id*** is read for each cell from row 1 to row ***&maxrows***. If the data retrieved has the attribute ***&attrcode*** (The ***get.cell*** function returns TRUE) it is stored in ***OldValue***, then, using the ***formula*** command, is concatenated to the '<XOUT>' text stored in the ***tag*** location and placed in ***NewValue*** before being output to the second sheet. If the strike-through attribute is not present the text is simply copied between ***OldValue*** and ***NewValue*** without the concatenation. After the last row has been reached the process halts and the DDE buffer is flushed clear. The last ***put*** statement runs the macro just created (***macro1***) using the second new sheet as output.

As this section of the program runs the changes being made to the Excel session appear 'live' on the screen. When this stage completes the ***xlmacro*** file must be closed using:

```
filename xlmacro clear;
```

Now the second worksheet has been populated with the copy of the input sheet data but with the struck out lines prefixed with '<XOUT>' all that remains is to identify the struck-out rows and select only the non struck-out rows. The following code is very similar to the ***original*** DATA step in that it creates a dataset of the first column (Column A) contents, but this time the data will include the '<XOUT>' tags (Step 3 in the diagram).

```
filename markfile dde "excel|&sheet_id!r1c1:r&maxrows.c1" notab lrecl=&maxrecl;

data marked;
  length col_a $&maxrecl;
  infile markfile dsd missover;
  input col_a;
run;

filename markfile clear;
```

This completes all the tasks involving the Excel application through DDE, but it must now be closed, and without saving the workbook so the original data is left unchanged. This is accomplished with the following code:

```
data _null_;
  file sas2xl;
  x=sleep(&delay);
  put '[error("false")]';
  put "[file.close()]" ;
  put "[quit()]" ;
run;

filename sas2xl clear;

data _null_;
  rc=sleep(&delay);
run;
```

Once again the delay is needed to allow MS Excel to perform the functions without SAS getting too far ahead. The '**[error("false")]**' ***put*** statement is to prevent the user dialog box 'Save changes Yes/No/Cancel' from being displayed and waiting for input from the user terminal.

The final task (Step 4 in the diagram), is to merge the ***marked*** data is with the ***original*** data by the first column, Column A, dropping observations with the tagged struck-through Column A values.

```
proc sort data=marked;
  by col_a;
run;

proc sort data=original;
  by col_a;
run;

data final;
  merge marked(in=a) original(in=b);
  by col_a;
  If a and b;
run;
```

Note: The ***in*** variables ***a*** and ***b*** may be used to include instead of exclude as required, by specifying:

```
if a and not b;
```

ATTRIBUTE CODES

The attribute code (attrcode) of 23 is the code for strikethrough font for the cell or the first character of the text in a cell. Bold text is selected with a code of 20, italic text with 21, and underlined text with 22. When any of these particular values are used with the VB function get.cell, the function returns a Boolean TRUE or FALSE result. Note: Two or more of these attributes may be present for a given cell, such as bold and italic, in which case setting ***attrcode*** to 20 *or* 21 will cause it to be tagged.

IDENTIFYING ROW COLOR

The Visual Basic get.cell command can be used with different attribute codes to determine the text (foreground) or cell (background) color, useful when rows are highlighted or shaded. For foreground color the attribute code is 64 and for background color the code is 63, but this time the get.cell function returns the numeric code for the color found instead of just TRUE or FALSE, hence the VB get.cell function must be tested against the color code for being TRUE or FALSE. For example:

```
get.cell(&attrcode,CurrentCell)=&color
```

where ***&attrcode*** is code 64 or 63 accordingly and ***&color*** is the numeric code for the particular color. The most commonly used colors and their codes are listed in the table below:

Color	Code
White	2
Gray 25%	15
Gray 40%	48
Gray 50%	16
Gray 80%	56
Black	1
Cyan Highlight	37
Green Highlight	35
Yellow Highlight	36
Magenta	13
Blue	5
Green	10
Yellow	6
Orange	46
Red	3

Table 1. List of Colors and Attribute Codes

NOTES

The column being marked (Column A in the above example) must have distinct values, so there are no repeats of ***by*** variables in the merge.

The Excel application must be closed prior to running this code. When Excel is opened using DDE it opens the sheet as it was last saved, including any sub-setting of a column contents.

A long enough input buffer (***&maxlen*** in the above example) must be specified to read in all of the longest line of original text without truncation.

The ranges of columns and rows (rnn:cnn) must be at least large enough to hold all of the input data, even if this original data is being sub-setted by ***if-then*** or a ***where*** clause. If the first row of the sheet contains titles this must be skipped by specifying the second row ‘r2’ instead of the first row as the top row of the range.

The above code has been tested on Windows 7 with Microsoft Office 2006 and later. Earlier versions of Windows or Microsoft Office may not be compatible.

SOURCES OF INFORMATION

All of the above information is derived from the author’s own experience, the SAS Institute and Microsoft corporation on-line support services, and informal on-line discussion threads.

CONTACT INFORMATION

Timothy J Harrington
Dataceutics Inc.
1600 Medical Drive
Suite 300
Pottstown PA 19464

www.dataceutics.com

Work Phone: 610 - 970 - 2333
FAX: 610 - 970 - 4884

Email: BritishCWhizz@Chartermi.net